

Interactive Isogeometric Volume Visualization with Pixel-Accurate Geometry

Franz G. Fuchs, Jon M. Hjelmervik

Abstract—A recent development, called isogeometric analysis, provides a unified approach for design, analysis and optimization of functional products in industry. Traditional volume rendering methods for inspecting the results from the numerical simulations cannot be applied directly to isogeometric models. We present a novel approach for interactive visualization of isogeometric analysis results, ensuring correct, i.e., pixel-accurate geometry of the volume including its bounding surfaces. The entire OpenGL pipeline is used in a multi-stage algorithm leveraging techniques from surface rendering, order-independent transparency, as well as theory and numerical methods for ordinary differential equations. We showcase the efficiency of our approach on different models relevant to industry, ranging from quality inspection of the parametrization of the geometry, to stress analysis in linear elasticity, to visualization of computational fluid dynamics results.

Index Terms—Volume visualization, Isogeometric analysis, Splines, Roots of Nonlinear Equations, Ordinary Differential Equations, GPU, Rendering

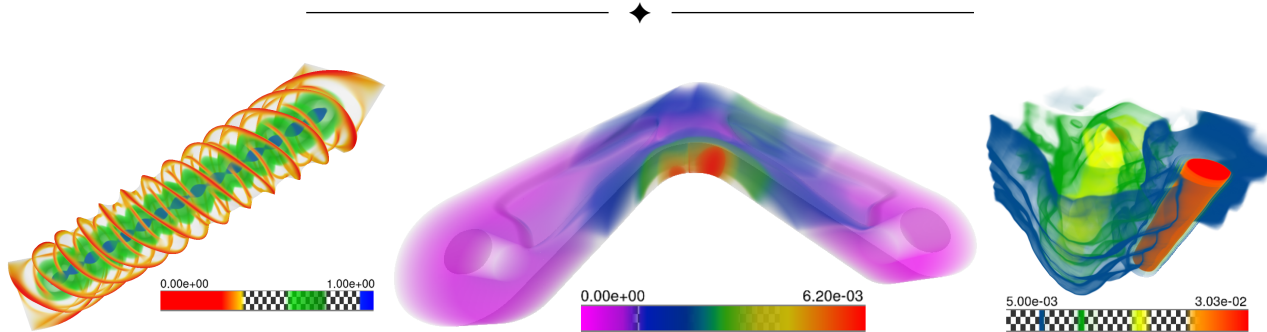


Fig. 1: Examples for isogeometric volume visualization in industry. Left to right: Twisted bar showing quality of parametrization; Industrial demonstrator model from TERRIFIC project showing von Mises stress of the bent model; Backstep flow from a computational fluid dynamics simulation showing turbulent viscosity. The bars show the colors that are assigned to the values of the scalar field, where a checkerboard pattern indicates transparent regions.

1 INTRODUCTION

Classic volume rendering is a method to display a two-dimensional projection of a three-dimensional scalar field that is discretely sampled on a Cartesian grid. In order to achieve this, a model for radiative transfer is used to describe absorption and emission of light along view-rays. This article extends classic volume rendering to isogeometric volumes, where both geometry and scalar field are given in terms of splines (NURBS, B-splines, etc.). We present a novel method for direct, interactive rendering of isogeometric models. The efficiency and applicability of the proposed isogeometric volume rendering method is showcased in three different application areas relevant to industry, see Fig. 1.

These types of models stem from isogeometric analysis (IGA), a recent development proposed by Hughes

et al. [4] for the analysis of physical phenomena governed by partial differential equations. IGA provides the integration of design and analysis by using a common representation for computer aided design (CAD) and finite element methods (FEM). This eliminates the conversion step between CAD and FEM, which is estimated to take up to 80% of the overall analysis time for complex designs [4].

The pipeline for design, analysis and optimization of functional products is depicted in Fig. 2. Visualization is used in all the stages; for quality inspection of the geometry, for studying the results of the numerical analysis, and for marketing purposes. It is therefore increasingly important to offer visualization techniques that are reliable, informative and visually pleasing. A main advantage of IGA is that it enables the direct feedback from numerical analysis results to the CAD model. However, for that process to work efficiently, it is essential to be able to interactively inspect the results from the numerical analysis stage.

The geometry of an isogeometric volume is given

• SINTEF ICT, Forskningsveien 1, N-0314 Oslo, Norway
E-mail: franzgeorgfuchs@gmail.com, jon.hjelmervik@sintef.no

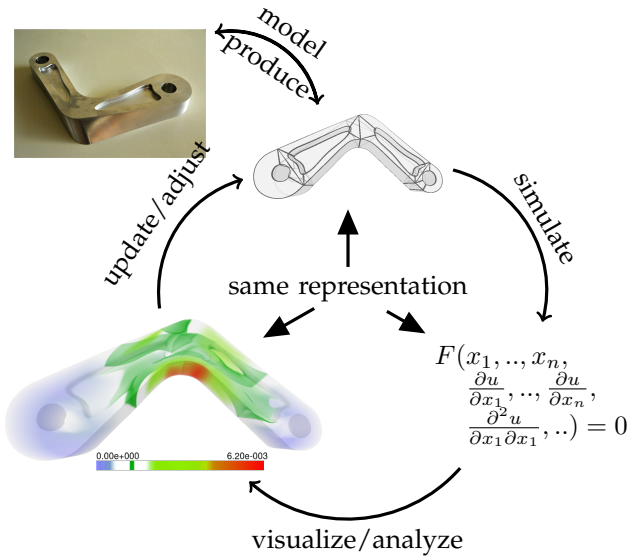


Fig. 2: Information flow of an isogeometric object.

by a spline ϕ , mapping each point of the parameter domain $P \subset \mathbb{R}^3$ to a point in the geometry domain $G \subset \mathbb{R}^3$, see Fig. 3. In addition, a second spline ρ is defined on the parameter domain P , describing a physical value such as density, displacement, temperature. The spline ρ can be scalar- or vector-valued and comes from a numerical simulation of a physical phenomenon.

In this paper we restrict our attention to B-splines. In three dimensions a B-spline of degree p has the form

$$S(u, v, w) = \sum_{i=1}^l K_i^p(u) \sum_{j=1}^m L_j^p(v) \sum_{k=1}^n M_k^p(w) C_{i,j,k}, \quad (1)$$

where $C_{i,j,k} \in \mathbb{R}^d$ are the control points defined over a set of non-decreasing knot vectors $U = \{u_1, \dots, u_{l+p+1}\}$, $V = \{v_1, \dots, v_{m+p+1}\}$ and $W = \{w_1, \dots, w_{n+p+1}\}$. With K_i^p, L_j^p, M_k^p we denote the recursively defined i -th B-spline of degree p in the corresponding direction. For a scalar spline $\rho(u, v, w)$ the control points $C_{i,j,k}$ are scalar valued, and for the spline $\phi(u, v, w)$ describing the geometry the control points have values in \mathbb{R}^3 .

2 RELATED WORK

Scientific volume visualization techniques convey information about a scalar field defined on a given geometry. The techniques can be divided into the following approaches: simply rendering the bounding surfaces of the object; iso-surface extraction; and volume rendering. The main challenges for achieving interactive volume visualization in the setting of isogeometric volumes are that an explicit expression of the inverse function of the geometry is not available in general, and that sampling is computationally

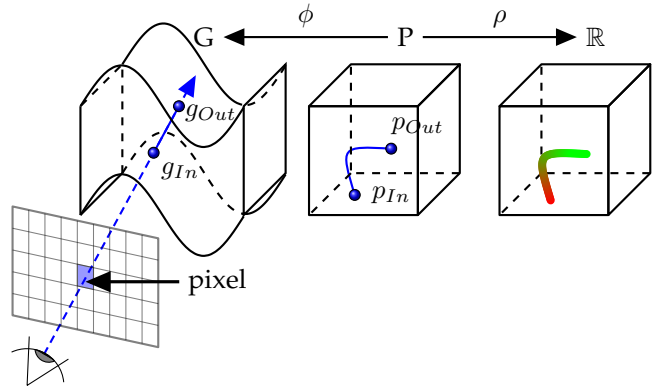


Fig. 3: Isogeometric volume rendering: ϕ describes the geometry, ρ defines a scalar field, both defined on P .

expensive due to the need for spline evaluation, i.e., piecewise polynomial functions.

The first approach for volume visualization is to render the scalar field on the outer surfaces of the isogeometric volume only. Although no information in the interior can be retrieved, this is a popular method due to its low computational effort. Methods based on ray-casting parametric polynomial surfaces are a well-studied but challenging problem, see e.g. Kajiya [13]. B-spline surfaces can be rendered as piecewise algebraic surfaces, see for instance Loop and Blinn [16]. But finding corresponding scalar field values becomes difficult, because algebraic surfaces are not parametrized. An alternative to ray-casting surfaces is rasterization, in particular with the recent introduction of the tessellation shader stage in graphics processing units (GPUs). A GPU-based two-pass algorithm for pixel-accurate rendering of B-spline surfaces was presented by Yeo et al. [26]: The first pass determines a sufficient tessellation level for each patch; during the second pass the surface is actually tessellated. An alternative method which is also pixel-accurate, is presented by Hjelmervik [11], where bounds on the second order derivatives decide sufficient tessellation levels, without querying neighboring patches, allowing a single-pass algorithm.

The second approach is to display iso-surfaces, i.e., surfaces where the scalar field has a particular value. If the scalar field is sampled discretely over a regular grid, the marching cubes algorithm (see Lorensen and Klein [17]) provides an efficient implementation; see e.g., Dyken et al. [7] for an implementation on the GPU. However, for isogeometric models, where both the geometry and the scalar field are given by spline functions, the marching cubes algorithm cannot be applied directly. Martin and Cohen [19] provide an algorithm for iso-surface extraction in the setting of isogeometric volumes. The method consists of iteratively dividing the model into a set of Bézier volumes. When those volumes are sufficiently simple, the iso-surfaces are given as the root of a function and

the Newton-Raphson method is used to approximate the surfaces. This framework has been realized as a CPU-based parallel implementation (see Martin et al. [18]). The reported timings between 1 and 7 frames per second (FPS) on a cluster, make the approach unsuitable for interactive visualization purposes.

Recently, Schollmeyer and Fröhlich [23] presented a GPU-based multi-pass visualization technique for direct iso-surface ray casting of NURBS-based isogeometric volumes. The first pass generates a list of ray intervals which potentially contain intersections with the faces of each Bézier cell. After applying culling and depth-sorting, this list is used to generate ray-surface intersections in the second pass. The ray-surface intersections are given by the roots of a system of nonlinear equations in the third pass. An elaborate root isolating method is applied to find all ray-surface intersections. A GPU-based implementation shows interactive volume visualization results of their method.

Another approach is to model the scalar field as a participating medium, where a modifiable transfer function specifies how field values are mapped to emitted color and transparency. If the field consists of discrete samples over a regular grid, an abundance of results is available, see e.g. Levoy [15] for an early example or Engel et al. [8] for an overview. In order to make use of existing standard volume rendering methods, one possibility is to precompute a "voxelized" version of the isogeometric model: Taking the geometry into account, one can store the values of the scalar field in a texture (the algorithm proposed in this article can readily be used for that). However, there are several draw-backs of this approach: Firstly, as illustrated in Fig. 4, it is difficult to represent the outer surfaces of the isogeometric volume using a voxel grid. Isogeometric objects typically have a smooth outer surface, representing a sharp transition between where a scalar field is defined and the outside. Standard volume rendering will therefore typically lead to spurious block-like structures. Secondly, volume rendering of a voxelized model will (tri-) linearly interpolated the sample points. A linear interpolation $p_1(x)$ of a function $f(x)$ between two sample points a, b has the following (optimal) error bound $|f(x) - p_1(x)| \leq \frac{(b-a)^2}{8} \max_{x \in [a,b]} \|f''(x)\|$, see e.g., [11]. A similar bound holds in 2 and 3 dimensions. In IGA the scalar field is given by $f(x, y, z) = \rho(\phi^{-1}(x, y, z))$, where ρ , and ϕ are spline functions (see Fig. 3). This means that a point (such as the red point in Fig. 4 (d)) that is (tri-) linearly interpolated between sample points can have an arbitrarily large approximation error, depending on the second order derivatives of $f(x, y, z)$. We refer also to Fig. 13 showing how this affects render quality. As a consequence, a voxelized version needs a potentially very high number of voxels in order to represent the scalar field accurately. The high demand on GPU memory decreases the efficiency of standard volume

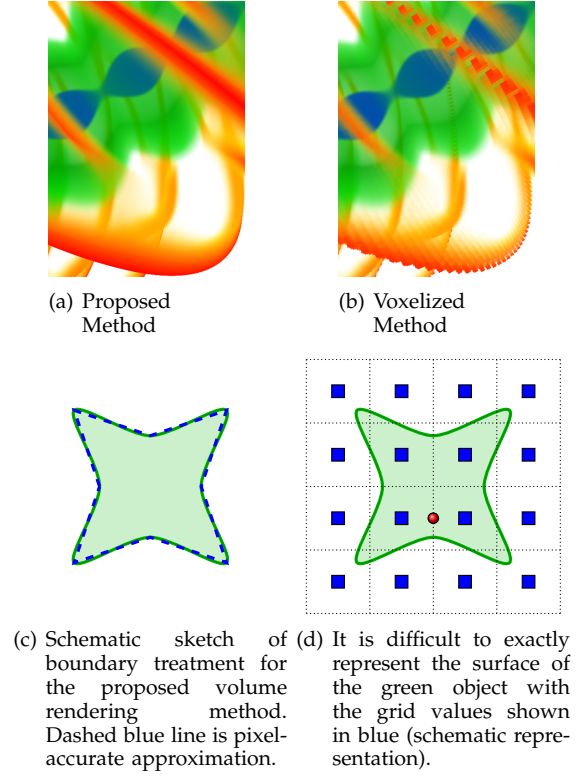


Fig. 4: The proposed method leads to pixel-accurate outer surfaces, while standard volume rendering of a voxelized version of the model is prone to show block-like structures at the outer surfaces. The red sample point in (d) will be (tri-) linearly interpolated.

rendering algorithms, see Section 6. Of course, more sophisticated methods with adaptively chosen sample points could be used to represent the scalar field. However, since a main reason for the introduction of IGA was to enable geometrically exact representation of geometry, this advantage should not be lost in the visualization stage, see Fig. 2. Exact geometry is desired by designers and analyzers alike.

An algorithm for direct volume rendering for freeform volumes was presented by Chang et al. [2]. The method consists of subdividing B-spline volumes into a set of Bézier volumes until the geometry of the volumes is monotone. Those volumes are then depth-sorted, and scan-converted, allowing direct blending. The algorithm reaches approximately 5 FPS on a mini-supercomputer with 8192 processors. Martin and Cohen [19] outline an algorithm based on finding the roots of a function with the Newton-Raphson method. However, to the best of our knowledge, there is no implementation of this algorithm for isogeometric models.

In [14] Kurzion and Yagel present a method for visualizing deformed two- and three-dimensional models. Instead of deforming the objects themselves, the geometry is given by so-called deflectors that bend

the rays used to render the scene. However, in contrast to isogeometric volumes, the geometry is given explicitly.

In this paper we present a flexible framework for volumetric visualization based on volume rendering, allowing visualization of scalar fields as well as derived properties such as parametrization quality or mechanical stress. Our approach consists of several stages, leveraging the strengths of existing algorithms where possible. Several features of our approach are novel:

- 1) We provide an extremely robust and efficient algorithm for determining view-ray intersections with the surfaces of the volume, see Section 4.1. This is achieved by reformulating the original problem of finding zeros of a function, to be a problem related to approximation of surfaces.
- 2) We devise novel approaches for pixel-accurate approximation of the preimage (i.e., the inverse) of the view-ray in the parameter space, suitable for efficient implementation on modern GPUs, see Section 5. In addition to an algorithm based on approximating zeros of a function, we provide an alternative based on ordinary differential equations (ODEs).
- 3) Degenerate cases of the parametrization of the geometry are treated in a suitable way, further increasing the robustness of our approach, see Section 5.4.

We would like to point the reader to [5] for a good introduction to numerical methods for ODEs as well as [3] for a mathematical introduction to fluid mechanics.

The rest of the paper is organized as follows: Section 3 provides the necessary background for volume rendering of isogeometric models, followed by a description of our approach in Section 4. Then, novel algorithms enabling geometrically pixel-accurate sampling of the volume render integral (2) are described in Section 5. Finally, we present applications and provide details of the performance of the implementation of the overall algorithm in Section 6, and a conclusion in Section 7.

3 VOLUME RENDERING FOR ISOGEO-METRIC MODELS

In this article we present an algorithm for volume rendering based on tracing view-rays through the volume from an imaginary observer. If such a view-ray intersects the object one obtains the color for the pixel of the screen by evaluating an integral describing the accumulated radiance along the ray. For a more detailed description of well established techniques for volume rendering see Engel et al. [8], Jensen [12] and references therein.

3.1 Continuous Model

When taking both emission and absorption into account, the accumulated radiance I_λ for wave length λ along a view-ray $\gamma : \mathbb{R} \rightarrow \mathbb{R}^3$ is given by the so-called volume render integral

$$I_\lambda(t) = I_\lambda(0)T_\lambda(0, t) + \int_{\gamma|_{[0, t]}} \sigma_\lambda(s)T_\lambda(s, t)ds, \quad (2)$$

where \int_γ denotes the line integral. The function $\sigma_\lambda(s)$ specifies emission, and $T_\lambda(s, t)$ specifies absorption (from s to t) of light with the wave length λ . In applications, one typically uses three groups of wave lengths representing red, green, and blue. The emission and absorption, defined by a so-called transfer function, depend on the value of the scalar field ρ .

A major difference to classic volume rendering is that the geometry is no longer trivial. For isogeometric models, both the spline $\phi(u, v, w)$ describing the geometry as well as the scalar field $\rho(u, v, w)$, are defined on the same parameter domain P , see Fig. 3. This means that the value of the scalar field along the view-ray $\gamma(s)$ is given by

$$\rho_\gamma(s) := \rho(\phi^{-1}(\gamma(s))). \quad (3)$$

If not otherwise stated, ϕ is assumed to be bijective. As mentioned before, both emission and absorption in the volume render integral (2) are functions of the values of the scalar field, i.e.,

$$\sigma(s) = \sigma(\rho_\gamma(s)), \quad T(s, t) = T(\rho_\gamma(s), \rho_\gamma(t)). \quad (4)$$

As a consequence, if ϕ is not linear, the value of the scalar field along the straight view-ray in the geometry domain G , is obtained along a (not straight) curve in the parameter domain P .

3.2 Numerical Approximation

In general, neither the inverse of the spline ϕ (see Equation (3)) nor the volume render integral (2) itself have closed-form solutions. Therefore, the solutions have to be approximated and the overall error will consist of different sources due to

- numerical quadrature of integral (2) (depending on number of sample points and their location), and
- numerical approximation of the preimage of these sample points along the view-ray (method depended).

For interactive applications, a compromise between performance and accuracy must be found. The numerical quadrature of the volume render integral is described in Section 3.2.1. The approximation of the inverse is specific to isogeometric models, and the main contribution of this paper is dedicated to it (sections 4 and 5). A definition of the requirement for the accuracy of the approximation of the inverse is given in Section 3.2.2.

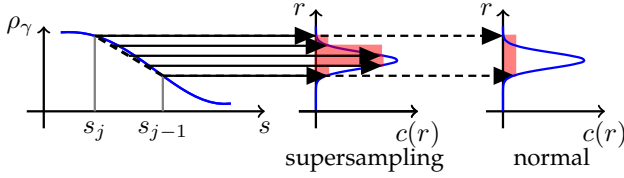


Fig. 5: Supersampling: Dense sampling of high frequencies of the transfer function c can improve the approximation (shown in red) of the volume render integral (2) without further evaluations of ρ_γ .

3.2.1 Quadrature of volume render integral

Discretizations of Equation (2) are based on splitting the integral into intervals. Efficient implementations on GPUs are so-called compositing schemes where color and opacity is accumulated iteratively. Front-to-back compositing for the accumulated radiance $C_{dst} = (I_r, I_g, I_b)^T$, and the accumulated opacity $\alpha_{dst} = (1 - T_{dst})$ is given by Algorithm 1.

Algorithm 1 Front-to-back compositing

```

 $T \leftarrow (1 - \alpha_{src})^{\Delta s_i / \xi}$ 
 $C_{dst} \leftarrow C_{dst} + (1 - T)(1 - \alpha_{dst})C_{src}$ 
 $\alpha_{dst} \leftarrow \alpha_{dst} + (1 - T)(1 - \alpha_{dst})$ 

```

Here, Δs_i is the varying ray segment length and ξ is a standard length. Furthermore, C_{src} and α_{src} are given by the transfer function through Equation (4).

In many application areas transfer functions contain high frequency components, dictating a high sampling rate (Nyquist rate). One method is *oversampling*, i.e., introducing additional sampling points, although the underlying scalar field is approximately linear. Evaluating the scalar function in the setting of isogeometric volume rendering is a time-intensive operation as it means computing (1).

To avoid oversampling a common technique is *pre-integration* (see e.g., [9], [22]), which is based on calculating the volume render integral for pairs of sample values in advance. Although this approach can successfully be applied in many cases, it has the flaw that it only works for equidistant sample points, because the volume render integral (2) is nonlinear.

To avoid oversampling, but account for the non-linearity of I , we use the following technique called *supersampling*. In contrast to pre-integration, supersampling uses a dense quadrature of the volume render integral assuming linearity of the scalar field between two sample points $\rho_\gamma(s_j), \rho_\gamma(s_{j-1})$ (not assuming linearity of the volume render integral between sample points), see also Fig. 5.

Although this approach also has its weaknesses, it is easy to implement, and it successfully captures high frequencies of the transfer function. This increases the image quality by reducing wood-grain artifacts, while

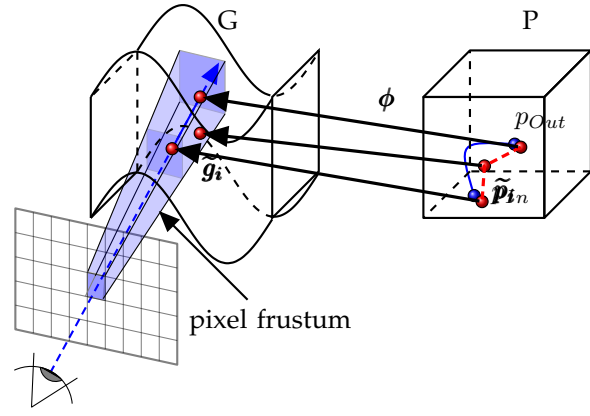


Fig. 6: Pixel-accuracy: The preimage of the view-ray (blue curve in P) is approximated by the red points \tilde{p}_i in P. The image $\tilde{g}_i = \phi(\tilde{p}_i)$ should lie inside the pixel frustum, shown as a blue box in G, and have the correct depth order.

avoiding computationally expensive evaluations of the spline function ρ and ϕ .

3.2.2 Pixel-Accurate Rendering of Geometry

In order to display the correct geometry of the isogeometric object, the approximation of the inverse of the geometry spline ϕ has to meet the following requirement, see Fig. 6 for an illustration. Following [11], [26], we define the following.

Definition 1 (Pixel-Accurate Approximation of Geometry). Let $\gamma(s)$ be the view-ray for a given pixel on the screen. The points $\tilde{p}_i, 1 \leq i \leq n$ in the parameter domain P are called pixel accurate sample points, if the following two requirements are fulfilled.

- All points \tilde{p}_i must project into the pixel of the view-ray ("parametric accuracy"):

$$\Delta P := 2 \left\| \pi_s(\phi(\tilde{p}_i)) - \begin{bmatrix} x \\ y \end{bmatrix} \right\|_\infty \leq 1, \quad (5)$$

where $\begin{bmatrix} x \\ y \end{bmatrix}$ is the pixel's center and $\pi_s : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ is the projection to the screen.

- All points \tilde{p}_i must have correct depth ordering along the view-ray ("covering accuracy"). For $2 \leq i \leq n$:

$$\|\pi_\gamma(\phi(\tilde{p}_i)) - g_{eye}\|_2 \geq \|\pi_\gamma(\phi(\tilde{p}_{i-1})) - g_{eye}\|_2, \quad (6)$$

where g_{eye} is the position of the observer, and $\pi_\gamma : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ is the orthogonal projection onto the view-ray.

This definition allows for a variable sample distance.

4 APPROACH AND IMPLEMENTATION

We present a novel approach enabling interactive volume visualization of isogeometric models with pixel-accurate geometry. It is often necessary to partition the model in order to be able to model real-life features such as holes or to improve the quality of the

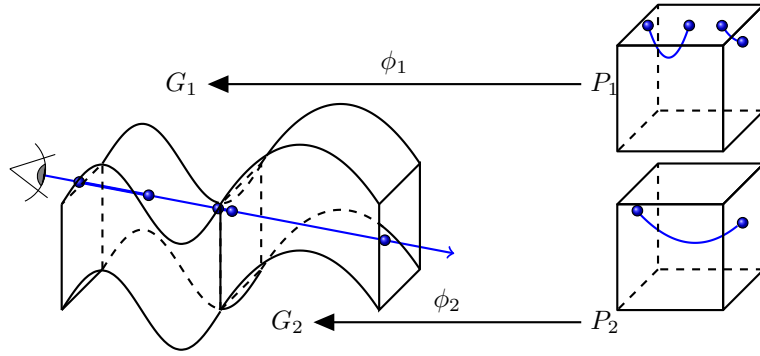


Fig. 7: This isogeometric model consists of two volume blocks G_1 and G_2 . Both non-convexity and the multi-block structure increase the number of possible intersections per view-ray.

parametrization. Therefore, isogeometric models often consist of a number of so-called *volume blocks* G_α , $1 \leq \alpha \leq a$, see Fig. 7. Each block has an corresponding spline $\phi_\alpha : P_\alpha \rightarrow G_\alpha$ defining the geometry, and scalar field $\rho_\alpha : P_\alpha \rightarrow \mathbb{R}$.

As described in Section 3, an important step for approximating the volume render integral (2) is to determine the intersections of the view-ray with the surfaces of the object. When designers create isogeometric objects they often collapse edges or align two faces, which leads to singularities of the Jacobian of ϕ_α on the boundary. Existing approaches often struggle with finding intersections, leading to computationally expensive algorithms. In order to devise an efficient and stable algorithm, our approach for volume-rendering consists of the following stages that are executed for every frame as part of the render pipeline.

- 1) *View-Ray intersections with surfaces*: We determine the intersections of all view-rays with the surfaces of the object by reinterpreting the problem as an approximation of surfaces, see Section 4.1.
- 2) *Depth-sorting of intersections*: Once all intersections are determined, they are sorted along each view-ray according to the distance from the origin (depth), see Section 4.2.
- 3) *Approximate Volume Render Integral*: For each pair of entry and exit points the volume render integral (2) is approximated, using *pixel-accurate* approximations of the *inverse of the view-rays*, see Section 5.

4.1 View-Ray Intersections with Surfaces

Computing the intersections between a ray and a spline surface can be a computationally expensive and unstable operation. Commonly, the problem is stated as finding the zeros of a function. However, the intersection problem can be restated as the problem of finding a view-dependent approximation of the surfaces of the object. Therefore, the rasterization process of GPUs can be used as a very efficient, parallel implementation of finding all ray-surface intersections of a

triangulation. Two alternative approaches to construct a view dependent triangulation using the hardware tessellator where recently presented by Yeo et al. [26] and by Hjelmervik [11]. Both methods are applicable in our setting, guaranteeing both water tightness and that the approximation error satisfies the requirements from Definition (1). Our implementation uses [11] since it provides a single-pass algorithm.

In our approach all boundary surfaces of all volume blocks G_α are rendered (6 surfaces per block) and the result is stored in a texture buffer.

4.2 Blockwise Depth-Sorting of Intersections

The first stage of our approach (see Section 4.1) will lead to an unordered list of intersections per view-ray. The number of intersections depends on view-angle, and the following two properties, depicted in Fig. 7:

- *Non-convex objects*: For non-convex objects, the view-ray can intersect the geometry multiple times per block, leading to multiple entry and exit points along the view-ray.
- *Multi-block objects*: Each volume block potentially adds further intersections.

Since modern GPUs allow atomic operations, our approach is based on a linked list (per pixel-location), which is populated in a single pass, as detailed in Yang et al. [25]. This method comes from a problem known as *order-independent transparency* in computer graphics, see e.g., Everitt [10], Carpenter and Li [1], and Myers and Bavoil [20].

This list is the basis for volume rendering in the next stage. In addition to the parameter value, each entry stores information about the depth, the block number, and the orientation (front facing or not). In order to produce a pixel-accurate result according to Definition (1), we choose pairs from the list of intersections according to Algorithm 2.

In IGA the surfaces of two neighboring blocks match exactly. The pixel-accurate rendering of the surfaces described in Section 4.1 will produce two entries in the list with approximately the same depth

Algorithm 2 Blockwise depth-sorting

- 1) Find p_{Out} by going through the list and choose entry
 - with depth value closest to screen,
 - that is not marked as used,
 - and is not front facing
 - 2) exit if the list is empty
 - 3) Find p_{In} by going through the list and choose entry
 - with depth value closest to screen,
 - that is not marked as used,
 - that is front facing,
 - and matches block number of p_{Out}
 - 4) render volume between p_{In} and p_{Out}
 - 5) mark p_{Out} and p_{In} as used
-

value. However, Algorithm 2 ensures that the found pairs correspond to the same block number.

5 PIXEL-ACCURATE INVERSE OF VIEW-RAYS

In the previous Section 4 we described how to obtain the intersections of the view-ray γ with the surfaces of the volume in the parameter domain P , providing pairs of entry and exit points (g_{In} and g_{Out}) for the compositing scheme. In order to approximate the volume render integral (2) pixel-accurately (see Definition (1)), approximations for sample points between g_{In} and g_{Out} have to be found.

This section describes two alternative methods for finding an approximation of the inverse of the function ϕ for all points on the view-ray between g_{In} and g_{Out} . If not otherwise mentioned, we will assume that each pair of entry and exit points

- 1) is given and pixel-accurate (ensured by Section 4.1),
- 2) belongs to only one volume-block, denoted by P_α (ensured by Section 4.2),
- 3) the line connecting the entry with the exit point does not intersect the boundary of block P_α (this case is described in Section 5.4), and
- 4) ϕ_α is a diffeomorphism with a continuous first derivative on the block P_α (see Section 5.4 for the case when J_ϕ is singular on the boundary ∂P_α).

We would like to point out that, under assumptions 1.-4., the problem of finding the inverse is well-posed in the sense of Hadamard, since ϕ_α is continuous, bijective and differentiable within the interior of each block. The aforementioned conditions on ϕ_α are reflected in the requirements for the numerical analysis performed on the isogeometric object.

The two alternative methods described in the following have certain advantages and disadvantages, discussed below. It is worth noting that neither

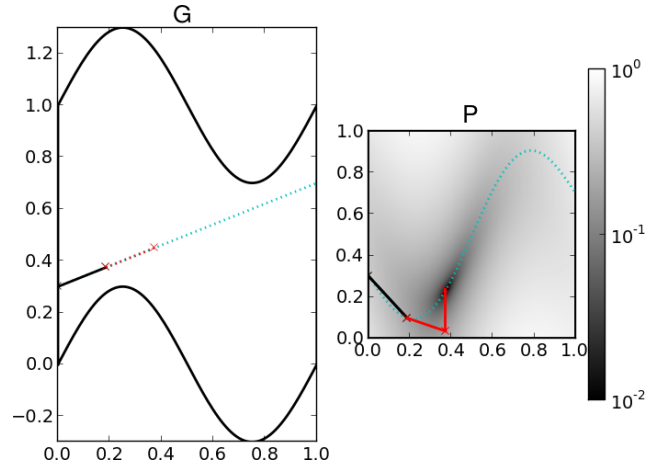


Fig. 8: One step in the root finding based method for a two dimensional example, where $\phi(x, y) = (x, 0.3 \sin(2\pi x))$. The red lines in the parameter domain P show how the Newton-Raphson method iteratively converges towards the zero of Equation (7). The underlain image in P shows the norm of F .

method is limited to the case where ϕ is a trivariate spline, but works as long as above assumptions on ϕ hold.

5.1 Root Finding Based Algorithm.

This first approach has been outlined by Martin and Cohen [19], but no practical details are discussed and, to the best of our knowledge, no implementation exists. We will provide a brief description of the algorithm, followed by a discussion of the approach.

Given $\phi : P \rightarrow G$ the problem is to find the preimage of sample points $g_i = g_{i-1} + \Delta s_i \frac{g_{Out} - g_{In}}{\|g_{Out} - g_{In}\|_{L^2}}$ along the view ray. Here, $g_0 = g_{In}$ and Δs_i is a (variable) sample distance. Mathematically, the point p_i that is the preimage of g_i is in the null space of the following function

$$F_{g_i} : P \rightarrow \mathbb{R}^3 : p \mapsto \phi(p) - g_i. \quad (7)$$

A standard method for finding approximations of the roots of function (7) is the Newton-Raphson method, given by

$$J_F(x_n)(x_{n+1} - x_n) = -F(x_n), \quad (8)$$

where J_F denotes the Jacobian matrix of F , and x_n are the approximations to the root of F . To solve the 3×3 linear system of equations (8) we employ the QR-algorithm, see e.g., [6]. See Fig. 8 for an example.

Note that for each iteration of (8), both ϕ and $J_F = J_\phi$ have to be evaluated at the same point. Since ϕ is a spline function the Jacobian can be evaluated cheaply by reusing calculations for ϕ .

The Newton-Raphson method converges quadratically for "good" starting points. The method can, however, fail in certain situations. We will address

each situation in the following for the problem at hand.

For a good argumentation let us note that a bijective ϕ induces a metric space (P, d_P) on the open parameter domain P of each block with

$$d_P(p_1, p_2) := \|\phi(p_1) - \phi(p_2)\|_{L^2}, \quad p_1, p_2 \in P. \quad (9)$$

According to our basic assumptions, Equation (7) has a unique solution and therefore a root of F is a root of the norm of F and vice versa. Since

$$\|F_{g_i}(p)\|_{L^2} = d_P(p, \phi^{-1}(g_i)), \quad (10)$$

the problem of finding the inverse is equivalent to finding the minimum distance to the preimage of g_i . As a consequence, F_{g_i} will not have horizontal asymptotes or local extrema/stationary points, and converge for all starting points in the interior of the block. Overshooting can be an issue in case of large distance between samples or where the geometry is complicated. Since we have a bounded domain P we need to clamp values to the range of the parameter domain, usually $[0, 1]^3$, and use the method of line search. We would also like to point out that since $J_F = J_\phi$ the Jacobian in Equation (8) is non-singular on $P \setminus \partial P$ due to our basic assumptions.

The Newton-Raphson method (8) needs a stopping criterion. In our case we require the method to be pixel-accurate, so we stop the iteration when the distance to the view-ray (given by $\|F_{g_i}(x_n)\|_{L^2}$) is less or equal to the minimum distance of the point g_i to the frustum boundary.

5.2 ODE Based Algorithm.

We will now describe a second approach based on the fact that the view-ray γ can be seen as an integral curve of a (first order linear) dynamical system, i.e., the solution of an ordinary differential equation (ODE). The idea is to directly work in the parameter domain by appropriately defining an ODE for which the image of the solution coincides with the original view-ray γ . This can be achieved by defining a vector field and numerically approximating the integral curve from the point where the ray enters the domain, see Fig. 9.

In order to describe our method we start by defining an ODE on the geometry domain through

$$g'(s) = V(g(s)), \quad g(0) = g_{In}. \quad (11)$$

We define the vector field to be

$$V(g) = V_{\parallel} + cV_{\perp}(g), \quad (12)$$

consisting of two components, where the constant c is the (positive) relative weight between the two components. The first component is a constant velocity parallel to the view-ray given by $V_{\parallel} = \frac{g_{Out} - g_{In}}{\|g_{Out} - g_{In}\|}$. The second component is a velocity perpendicular to

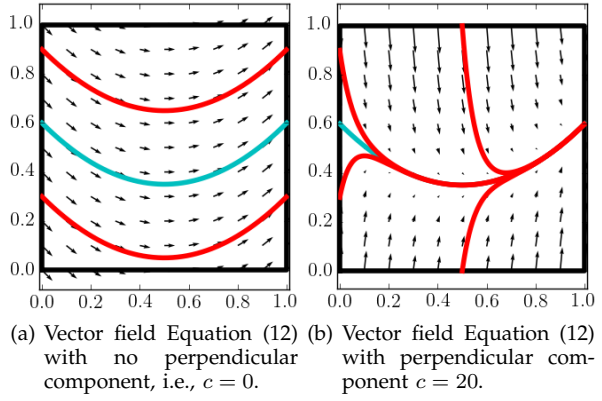


Fig. 9: The view-ray is shown in cyan, and streamlines for different start points are shown in red. The perpendicular component makes sure that the solution quickly converges towards the view ray.

the view-ray and depends on the signed distance to the view-ray, i.e.,

$$V_{\perp}(g) = (g_{In} - g) - \langle g_{In} - g, V_{\parallel} \rangle V_{\parallel}, \quad (13)$$

where $\langle \cdot, \cdot \rangle$ is the usual scalar product. The motivation for introducing the perpendicular component V_{\perp} is the following. The numerical method will inevitably introduce errors. The larger c the more will the numerical approximation be forced back to the exact solution, see Fig. 9 for an illustration.

The ODE (11) is a first order linear dynamical system, which can be rewritten in the standard form

$$g' = Ag + b, \quad \text{with} \quad A = c \begin{pmatrix} V_{\parallel,1}^2 - 1 & V_{\parallel,1}V_{\parallel,2} & V_{\parallel,1}V_{\parallel,3} \\ V_{\parallel,2}V_{\parallel,1} & V_{\parallel,2}^2 - 1 & V_{\parallel,2}V_{\parallel,3} \\ V_{\parallel,3}V_{\parallel,1} & V_{\parallel,3}V_{\parallel,2} & V_{\parallel,3}^2 - 1 \end{pmatrix}, \quad (14)$$

$$b = cg_{In} + (1 - c \langle g_{In}, V_{\parallel} \rangle) V_{\parallel}.$$

The dynamics are determined by the eigenstructure of A , see e.g., [21]. There are two negative eigenvalues $\lambda_{1,2} = -c$ and one zero eigenvalue $\lambda_3 = 0$ with corresponding eigenvector $v_3 = V_{\parallel}$. We can write a vector $u \in \mathbb{R}^3$ as the sum $u = u_{\perp} + u_{\parallel}$, where $\mathbb{R}^3 = M_{\perp} \oplus M_{\parallel} = \text{span}(v_1, v_2) \oplus \text{span}(v_3)$. Furthermore, there exists a unique vector $m \in M_{\perp}$ such that $Am = b_{\perp}$. The solution of Equation (14) is then

$$g(t) = e^{At}(g_{\parallel} + m) - m + tb_{\parallel}, \quad (15)$$

where $g_{\parallel}, b_{\parallel} \in M_{\parallel}$. This analysis shows that the view-ray is a solution and that all solutions (irrespective of the starting point $g(0)$) converge exponentially towards the view-ray.

Next we will define a vector field $W(p)$ such that the solution of the ODE on the parameter domain P

$$p'(s) = W(p(s)), \quad (s, p) \in \mathbb{R} \times [0, 1]^3, \quad (16)$$

$$p(0) = p_{In},$$

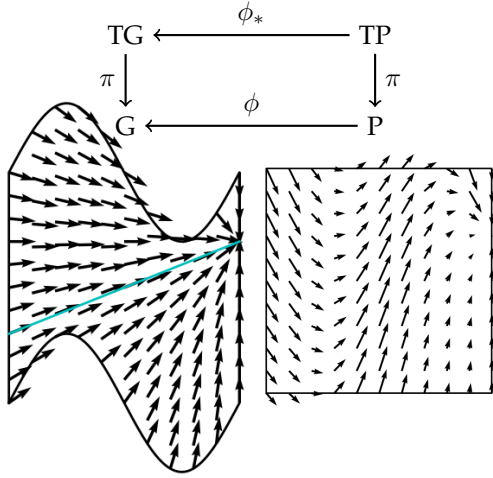


Fig. 10: Tangent bundle for $\phi(x, y) = (x, y + 0.3 \sin(2\pi x))$ with vector field defined on G and pulled back to P . Projection given by $\pi(\omega, v) = \omega$.

coincides with the preimage of the solution of Equation (11).

Using that ϕ is a differentiable map on the inside of P , the dual is given by

$$\phi_* : TP \rightarrow TG, (p, v) \mapsto (\phi(p), J_\phi(p)v), \quad (17)$$

where T denotes the tangent space, see e.g., Spivak [24]. We then have that the diagram shown in Fig. 10 commutes.

Since ϕ is a diffeomorphism on the inside of P , it follows that the dual has an inverse given by,

$$\phi_*^{-1} : (p, v) \mapsto (\phi^{-1}(p), (J_\phi(p))^{-1}v), \quad (18)$$

using the inverse function theorem. This can be used to "pull-back" the vector field $W(p)$ in Equation (16), by solving the following system of linear equations:

$$J_\phi(p)W(p) = V(\phi(p)), \quad (19)$$

Note that J_ϕ exists and is non-singular for all p in the inside of P . See Fig. 11 for an example of how the vector field becomes non-trivial in the parameter domain.

Since the vector field W is in general non-linear, we establish the following theorem.

Theorem 1 (Existence and Uniqueness). *Under the condition that ϕ is a diffeomorphism with a continuous Jacobian, there exists a unique solution to the initial value problem (16) (with (19)) that continues up to the boundary.*

Proof: According to Theorem 2.7 in [5] (based on the theorem of Picard-Lindelöf) it is enough to show that the right hand side of Equation (16) is continuous and locally Lipschitz-continuous. For every compact subset of $S \subset P$ there exists a constant C_ϕ , such that for all $v_1, v_2 \in \mathbb{R}^3$ and $p_1, p_2 \in S$

$$\|J_\phi^{-1}(p_1)v_1 - J_\phi^{-1}(p_2)v_2\| \leq C_\phi\|v_1 - v_2\|,$$

since ϕ has a continuous Jacobian. It is then easy to show that

$$\|W(p_1) - W(p_2)\| \leq C_\phi\|A\|_{op}L_\phi\|p_1 - p_2\|, \quad (20)$$

where $\|\cdot\|_{op}$ is the operator norm and L_ϕ is the Lipschitz constant of ϕ . \square

Solutions of the ODE (16) can be approximated using a wide variety of numerical methods, e.g., explicit Runge-Kutta methods (see [5]). The simplest scheme is the explicit Euler method given by

$$p(s^{n+1}) = p(s^n) + \Delta s^n W(p(s^n)), \quad (21)$$

where we use standard notation using superscripts indicating discrete "times" and $\Delta s^n = (s^{n+1} - s^n)$. Note that in each step we have to solve the linear system (19).

The stiffness index of Equation (14) is $L = \max_i(|\lambda_i|) = c$, see e.g., [5]. While larger c have the benefit of preventing the numerical approximation to deviate to far from the view-ray (see Fig. 11), the ODE will become increasingly stiff. This behavior will be inherited in the ODE on P as well. Explicit solvers, such as the one described in Equation (21), will suffer from impractically small Δs^n for large c . Generally, implicit schemes have larger stability regions. An *A-stable* method suitable for stiff equations is the *implicit Euler* scheme, given by

$$p(s^{n+1}) = p(s^n) + \Delta s^n W(p(s^{n+1})), \quad (22)$$

where a system of non-linear equations must be solved in each step. In order to achieve maximum stability, we do not solve for $p(s^{n+1})$ directly, but rather for the difference to the previous point, see e.g., [5]. Thus, we have to solve

$$G(z) = z - \Delta s^n W(z + p(s^n)) = 0, \quad (23)$$

numerically, for which we apply the Newton-Raphson method in order to approximate the solution. This method depends on the Jacobian of G , which can be approximated. However, in order to increase performance we avoid further evaluations of G and instead derive an exact expression for J_G . By rewriting Equation (23) as

$$J_\phi(z + p^n)G(z) = J_\phi(z + p^n)z - \Delta s^n V(\phi(z + p^n)), \quad (24)$$

where $p^n = p(s^n)$ and applying the Jacobian operator to both sides, we derive (using the chain- and product rule) the following linear system of equations

$$J_\phi(z + p^n)J_G(z) = H_\phi(z + p^n)(z - G(z)) + (\mathcal{I} - \Delta s J_V(\phi(z + p^n)))J_\phi(z + p^n). \quad (25)$$

Here, \mathcal{I} is the identity matrix and H_ϕ denotes the Hessian of ϕ , a tensor of order 3. Observe, that the spline ϕ along with its first and second order derivatives are evaluated at the same point, allowing for an efficient

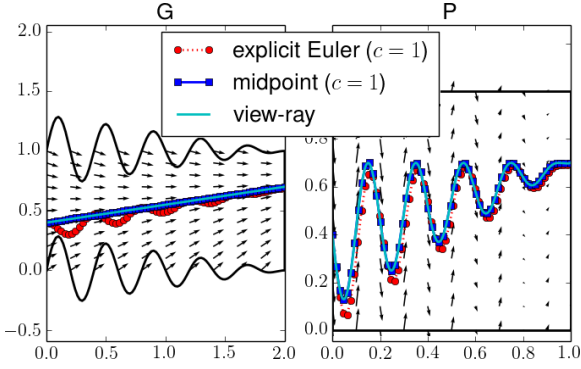


Fig. 11: With the ODE based method, the preimage of the view-ray in the geometry domain G is given by an integral curve in the parameter domain P .

calculation in a shader program. The Jacobian of the vector field V is the following constant matrix

$$J_V(g) = c(C - I), \quad (26)$$

where the i -th column of C is given by $V_{\parallel, i} V_{\parallel}$.

By solving the linear Equation (25) we can calculate the Jacobian matrix of G and use it for the Newton-Raphson method used for the implicit Euler method. It has the same matrix as the equation we need to solve to get the vector field W , see Equation (19). This means that, when using the QR-algorithm for solving both linear equations, an efficient algorithm can reuse Q and R for solving Equation (25).

5.3 Convergence study.

Fig. 11 shows a two dimensional test case given by $\phi(x, y) = (2x, y + 0.3(1 - x)\sin(10\pi x))$ with $p_{In} = (0, 0.3)$, $p_{Out} = (1, 0.7)$. In TABLE 1 we show how the different numerical methods converge to the exact solution. We use the following notation: (RK 1) explicit Euler method (first order), (IRK 1) implicit Euler method (first order), (RK 2) midpoint method (second order), (RK3) Kutta's 3rd order method, (RK4) classic 4th order method, (RK4 3/8) 3/8 rule (4th order), (RKF) Runge-Kutta-Fehlberg method (5th order), and (RF) root finding method. We observe that the ODE based algorithms converge with the expected order as the sample distance is reduced and the root finding based method reaches the given tolerance for all sample distances.

5.4 Degeneracies and Points Outside Domain.

There are two prominent cases for which the methods described in sections 5.1 and 5.2 need minor adjustments. The first case is when the line between g_{In} and g_{Out} intersects the boundary of the volume block. Although this is a rare case, it can happen that the approximation of the surfaces "misses" intersections in the tessellation of the geometry (described in Section 4.1). This case is shown in Fig. 12, where the

Δs	1.6e-02	7.8e-03	3.9e-03	2.0e-03
RK 1 (c=1)	6.2e-02	3.1e-02	1.6e-02	7.8e-03
IRK 1 (c=100)	5.3e-03	1.6e-03	8.1e-04	5.4e-04
RK 2 (c=1)	8.6e-04	2.1e-04	5.2e-05	1.3e-05
RK 3 (c=1)	1.5e-06	1.7e-07	2.1e-08	2.5e-09
RK 4 (c=1)	4.2e-07	3.0e-08	2.0e-09	1.3e-10
RK 4 3/8 (c=1)	1.7e-07	1.3e-08	8.8e-10	5.7e-11
RKF (c=1)	3.0e-08	9.5e-10	2.9e-11	9.2e-13
RF (tol = 1e-3)	9.8e-04	9.8e-04	5.2e-04	1.3e-04
RF (tol = 1e-14)	2.5e-16	2.5e-16	2.5e-16	2.5e-16

TABLE 1: For the case depicted in Fig. 11, the ODE based algorithms (for notation see Section 5.3) show the expected convergence rates, and the root finding based method (RF) reaches the given tolerance. The error is defined by $e_{L^\infty} = \max_i \|(p_{In} - \phi(p_i)) - < p_{In} - \phi(p_i), V_{\parallel} > V_{\parallel}\|_{L^2}$.

approximated surface (dashed black line in geometry domain) is still pixel-accurate, but the view-ray intersects the exact surface. In such a case, the Newton-Raphson method in both the implicit Euler method as well as the root finding based method will not converge, but repeatedly try to exit the parameter domain of the corresponding block. We detect such behavior and step along the boundary of the parameter domain until the view-ray is within the domain again. For the explicit Runge-Kutta methods we simply clamp the approximated solution values to remain in P . Observe, that the resulting approximation of the view-ray (seen in Fig. 12) is still pixel-accurate, since the approximation of the surface is guaranteed to be so.

The second case is when there are degeneracies of the spline ϕ along the boundary, see e.g., Fig. 14. Assume for instance that the Jacobian J_ϕ is singular at the entry point g_{In} . Since both the root finding method (Section 5.1) as well as the ODE based methods (Section 5.2) involve solving a system of linear equations with a singular matrix in that case (see Equations (8) and (19)), the only viable choice is to "shrink" the block in the following way. By choosing a new entry point $\tilde{g}_{In} = g_{In} + \delta \frac{p_{Out} - p_{In}}{\|p_{Out} - p_{In}\|_{L^2}}$, with a suitable $\delta > 0$. The new entry point \tilde{p}_{In} in the parameter domain P can be found with the root finding method with a different starting point for the iterations in the Newton-Raphson method, for instance

$$x_0 = p_{In} + \varepsilon \frac{p_{Out} - p_{In}}{\|p_{Out} - p_{In}\|_{L^2}}, \quad (27)$$

with an appropriately chosen ε . As can be seen in the middle of Fig. 1 this approach works well.

5.5 Cutting and Near Clip Planes

The root finding based method is well suited for realizing cutting planes. Before the final compositing, the depth-sorted list of intersections (from Section 4.2) is handled in the following way

- *Cutting planes:* The plane is given by a point $g_0 \in G$ and a normal n . If $(n \cdot g_{In} - g_0)$ and $(n \cdot g_{Out} - g_0)$

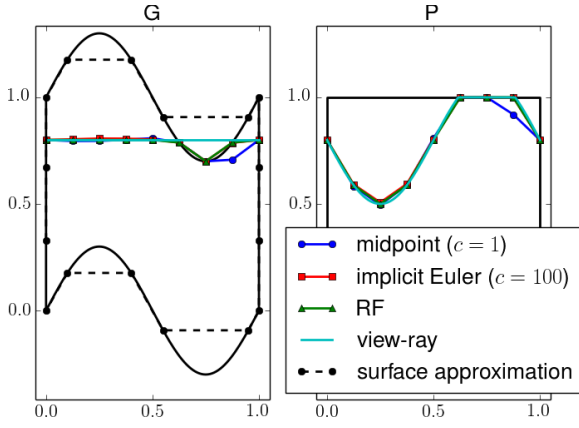


Fig. 12: When the view-ray intersects the boundary of a block due to a large pixel-frustum, both the root finding method (RF) and the methods based on ODEs still lead to a pixel-accurate approximation.

have opposite signs, the view-ray γ between g_{In} and g_{Out} intersects the cutting plane.

- *Near-Clip plane*: Since the rendering of the surfaces is water tight, an odd number of ray-surface intersections means that the near plane is inside the volume.

In both cases, trivial formulas determine the intersection point $g_* \in G$. Given g_* , we need to find the corresponding point $p_* \in P$ such that $\phi(p_*) - g_* = 0$. Thus, finding p_* is exactly solving Equation (7). Since this point can be quite far away from p_{In} or p_{Out} the root finding algorithm described in Section 5.1 is the best alternative and can readily be used.

6 APPLICATIONS AND PERFORMANCE

In order to benchmark the performance of the proposed methods, we present the three different application scenarios shown in Fig. 1, covering a wide range of possible applications. We apply the approach described in Sections 4 and 5 in each case, and compare it with standard volume rendering algorithms, where we have precomputed a voxelized version of the model. The resulting texture has 16 bit and uses the red channel for the scalar value and the green channel to encode if the voxel is inside the object or not. Of course, many optimization strategies are established in standard volume rendering, such as adaptive sampling rates, out of core algorithms, et cetera. However, to allow for a fair comparison, we only use an out of the box implementation without any optimizations. In all cases we measure the performance of our algorithm on an NVIDIA Titan GPU.

The proposed approach, allocates memory for the knots and the control points. In addition, a buffer is allocated for the linked list containing all view-ray intersections with the surfaces (see Section 4.2).

In order to measure how well the volume render integral (2) is approximated we measure the color

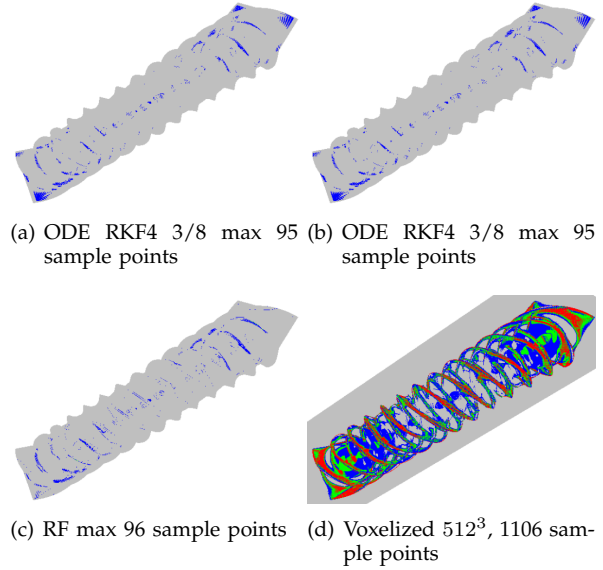


Fig. 13: Visualization of color difference to a reference solution. Each pixel is colored [grey, blue, green] if ΔE is at most [1,5,10]. If it is red, the color difference is greater than 10. ΔE is the CIEDE2000 definition of color difference. Standard volume rendering of a voxelized version shows large errors even though 10 times as many samples are used.

difference to a reference image with the norm “ ΔE ” as defined by the International Commission on Illumination (CIE) in 2000. In this metric $\Delta E = 1$ means a “just noticeable difference”. We will, however, be less strict and regard values of up to 5 to be acceptable.

We start with an application useful in the design phase of the geometry.

6.1 Parametrization Quality of Geometry

Before an analysis of a model can be carried out, the geometric shape has to be designed. Since the parametrization of the geometry ϕ is not unique, one wants to optimize the quality of the parametrization. A measure of the quality of the parametrization is given by

$$\rho = \frac{\det(J_\phi)}{\|J_\phi\|_F}, \quad (28)$$

where low values indicate that the geometry is (close to) degenerate. With this scalar field our method can be used as an inspection tool in the design phase, isolating potentially problematic areas. The parametrization quality (28) is calculated on the fly for each sample.

As an example we present a twisted bar, see left in Fig. 1. This model consists of only one volume block and the geometry is given by a quadratic B-spline with 425 control points. We compare standard ray-casting of the pre-computed voxelized method with the proposed methods on a screen resolution

(a) ODE based method RK4 3/8 (fourth order).

max(# S)	max(ΔP)	max(ΔE)	mean(ΔE)	var(ΔE)	surf [ms]	ray [ms]	tot [ms]
11	2.1	55.064	2.946	21.834	0.86	8.02	9.15
23	0.6	34.447	1.421	7.211	0.86	13.74	14.91
47	0.6	17.771	0.537	1.092	0.86	24.92	26.18
95	0.6	4.728	0.189	0.111	0.86	46.67	47.97
190	0.6	2.105	0.077	0.022	0.86	88.22	89.74
381	0.6	1.167	0.038	0.012	0.86	172.70	174.50
762	0.6	1.092	0.020	0.006	0.86	337.30	339.70
1525	0.6	0.930	0.009	0.002	0.86	663.50	666.50

(b) Voxelized method with texture size 227^3 .

max(# S)	max(ΔE)	mean(ΔE)	var(ΔE)	ray [ms]
70	79.283	6.609	157.129	1.00
139	79.302	6.111	137.939	1.58
277	78.049	5.616	118.935	2.60
553	77.923	5.278	106.801	4.57
1106	77.809	5.075	99.912	8.33
2211	77.726	4.962	96.138	15.38
4422	77.681	4.901	94.120	29.41
8844	75.615	4.865	92.794	55.56

(c) ODE based method RK2 (second order).

max(# S)	max(ΔP)	max(ΔE)	mean(ΔE)	var(ΔE)	surf [ms]	ray [ms]	tot [ms]
11	20.4	54.582	3.066	22.910	0.86	5.87	6.99
23	4.8	35.191	1.433	7.277	0.86	9.43	10.57
47	1.1	17.771	0.531	1.075	0.86	16.15	17.30
95	0.6	4.745	0.186	0.108	0.86	29.43	30.67
190	0.6	2.110	0.076	0.022	0.86	55.33	56.78
381	0.6	1.167	0.038	0.012	0.86	105.80	107.40
762	0.6	1.091	0.020	0.006	0.86	203.10	205.70
1525	0.6	0.929	0.009	0.002	0.86	408.50	411.10

(d) Voxelized method with texture size 341^3 .

max(# S)	max(ΔE)	mean(ΔE)	var(ΔE)	ray [ms]
70	79.213	5.760	126.757	1.11
139	78.936	5.046	99.949	1.78
277	77.219	4.449	80.760	2.92
553	74.998	4.033	67.321	5.08
1106	74.790	3.777	59.694	9.26
2211	74.753	3.638	55.761	17.24
4422	74.800	3.563	53.635	33.33
8844	74.612	3.523	52.441	62.50

(e) Root finding based (RF).

max(# S)	max(ΔP)	max(ΔE)	mean(ΔE)	var(ΔE)	surf [ms]	ray [ms]	tot [ms]
12	1.4	54.053	3.404	24.938	0.86	4.95	6.06
24	0.8	38.387	1.667	8.329	0.86	8.86	10.00
48	0.6	16.046	0.638	1.080	0.86	16.37	17.52
96	0.6	7.263	0.246	0.123	0.86	31.77	33.01
192	0.6	3.394	0.119	0.030	0.86	61.80	62.15
384	0.6	2.138	0.074	0.018	0.86	121.40	123.00
768	0.6	1.105	0.055	0.013	0.86	242.60	244.70
1536	0.6	0.962	0.035	0.006	0.86	484.50	487.10

(f) Voxelized method with texture size 512^3 .

max(# S)	max(ΔE)	mean(ΔE)	var(ΔE)	ray [ms]
70	75.592	5.140	105.246	1.54
139	74.997	4.245	73.714	2.65
277	74.956	3.581	55.287	4.03
553	74.624	3.133	43.332	6.41
1106	74.642	2.860	36.356	11.49
2211	72.952	2.701	32.473	21.28
4422	72.790	2.624	30.659	41.67
8844	72.414	2.585	29.749	83.33

TABLE 2: Statistics for the example shown left in Fig. 1 (parametrization quality for the twisted bar). max(# S): maximum number of sample points; max(ΔP): largest error of pixel-accuracy among all pixels of the object; max/mean/var(ΔE): largest/mean of/variance of the color difference of a reference image; surf [ms]: time for creating ray-surface intersections; ray [ms]: time for blockwise depth-sorting and volume rendering for the proposed methods, time for standard volume rendering for "voxelized" method (texture is precomputed); tot [ms]: total render time.

of 640×480 pixels. The first difference is that the proposed method uses 63 MB, and the voxelized method uses [67,178,528] MB for a texture size of $[227^3, 341^3, 512^3]$. For the proposed methods we can see in TABLE 2 that the parametric accuracy ΔP (as defined in Equation (5)), decreases to 0.6 as the number of samples increases. It does not decrease further because the $\Delta P = 0.6$ is already reached for the view-ray intersections on the surfaces. We can also see that the color difference ΔE decreases with the number of sample points.

For the standard volume rendering of the precomputed ("voxelized") version of the model, the notion of pixel-accuracy is not meaningful. Ultimately, one is interested in the color difference ΔE . TABLE 2 shows that ΔE decreases with the number of sample points (for the volume render integral). The color difference also decreases with increased texture size. Naturally, for the same number of sample points along the view-rays, the volume rendering of the voxelized model is much faster than the proposed methods. However, for a texture size of 512^3 with almost 9000 sample points, the maximum color difference is still around 72, and

the mean is larger than 2. We can see in Fig. 13 that the color difference is highest at the boundary of the model, but for the voxelized model even the interior points show values between 5 and 10, meaning a noticeable difference to the reference image.

Overall, TABLE 3 shows that the second order ODE based method performs best on this model. Higher order ODE based methods typically allow larger sample distances while still being pixel-accurate. In this example all ODE based methods, except the first order methods, have the same sample distance which is dictated by the volume render integral. Therefore the second order midpoint method is fastest.

6.2 Stress Analysis in Linear Elasticity

Structural analysis is an important application area for isogeometric analysis, where external forces lead to a deformation of the object given in the form of a so-called displacement field $u : P \rightarrow \mathbb{R}^3$. The stress due to deformation is then calculated by

$$\rho = \left((\sigma_{11} - \sigma_{22}) + (\sigma_{22} - \sigma_{33}) + (\sigma_{33} - \sigma_{11}) + 6(\sigma_{12}^2 + \sigma_{23}^2 + \sigma_{31}^2) \right) / 2. \quad (29)$$

(a) Parametrization quality for the twisted bar, see left in Fig. 1. (b) Von Mises stress for TERRIFIC model, see middle in Fig. 1. (c) Backstep Flow from RANS simulation, see right in Fig. 1

method	tot [ms]	method	tot [ms]	method	tot [ms]
RK 1 (c=1)	332	RK 1 (c=1)	354	RK 1 (c=1)	41
IRK 1 (c=100)	204	IRK 1 (c=100)	321	IRK 1 (c=100)	76
RK 2 (c=1)	31	RK 2 (c=1)	78	RK 2 (c=1)	52
RK 3 (c=1)	37	RK 3 (c=1)	93	RK 3 (c=1)	67
RK 4 (c=1)	46	RK 4 (c=1)	108	RK 4 (c=1)	83
RK 4 3/8 (c=1)	48	RK 4 3/8 (c=1)	109	RK 4 3/8 (c=1)	82
RKF (c=1)	68	RKF (c=1)	137	RKF (c=1)	111
RF	47	RF	73	RF	62

TABLE 3: Comparison of the performance of the proposed methods for visualization of different models on an NVIDIA Titan GPU and a screen resolution of 640×480 . All methods use the largest (uniform) sample distance of the volume render integral, but are at the same time pixel accurate, i.e., $\Delta P \leq 1$ (as defined in Equation (5)) and the color difference to a reference image is $\Delta E \leq 5$.

The so-called strain tensor is $\sigma = \frac{1}{2}(J_{u \circ \phi^{-1}} + J_{u \circ \phi^{-1}}^T)$, where $J_{u \circ \phi^{-1}}(g)$ is the solution of

$$[J_{\phi}(p)]^T J_{u \circ \phi^{-1}}(g) = [J_u(p)]^T.$$

As in the previous example, all those expressions are calculated on the fly for each sample of the scalar field.

In the middle of Fig. 1 we present the results for the linear elasticity simulation from the TERRIFIC project. Both geometry ϕ and deformation u are cubic B-splines and the model consists of 15 volume blocks with 2484 control points. All the proposed methods work well also in this case where there are some degeneracies along the boundaries, see Fig. 14. As TABLE 3 (b) indicates, the root finding based method and the second order midpoint method have the fastest render times.

6.3 Computational Fluid Dynamics

Another important application of isogeometric analysis is computational fluid dynamics (CFD). On the right of Fig. 1 we present a visualization of an approximation of the solution of the Reynolds-averaged Navier-Stokes (RANS) equations for a backstep flow. The scalar field ρ represents turbulent viscosity and comes directly from the simulation. The model uses quadratic B-splines to represent both the geometry and the scalar field and consists of 140 blocks with 888642 control points. As can be seen from TABLE 3 (c) the explicit Euler method has the fastest render time, followed by the second order midpoint method and the root finding based method. Since the geometry and the scalar field is close to linear, the first order ODE solver is the most efficient compared with higher order ODE solvers.

7 CONCLUSION

The presented approach allows interactive inspection of volumetric models used in isogeometric analysis. In the spirit of isogeometry, the algorithms operate directly on the spline models and therefore demand

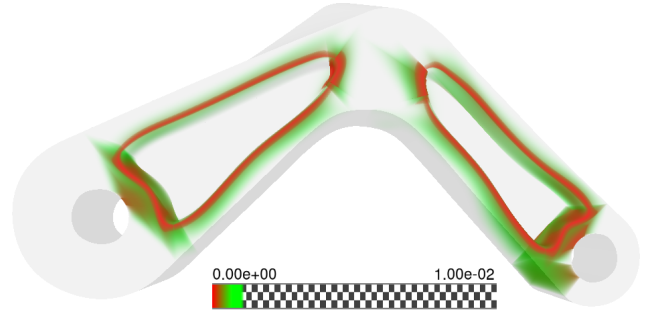


Fig. 14: Quality of parametrization (see Equation (28)) for the model from the TERRIFIC project see middle of Fig. 1. Low values shown in red indicate areas with a problematic parametrization. The checkerboard pattern indicates alpha values less than 1.

very little GPU memory. The proposed algorithms enable pixel-accurate geometry of both surfaces and volume irrespective of the zoom level, making it an asset during the design, analysis and marketing phase. We applied our approach in three use cases relevant to industry showing good performance at interactive frame rates.

In the future we plan to develop algorithms that automatically choose a sample distance that ensures pixel-accuracy. In addition, we seek to increase the efficiency of the presented methods by developing algorithms for adaptive sampling, as well as exploring methods for automatically choosing the order of the ODE based methods.

REFERENCES

- [1] Loren Carpenter and Star Trek II. The a-buffer, an antialiased hidden surface method. In *Computer Graphics*, pages 103–108, 1984.
- [2] Y.-K. Chang, A. P. Rockwood, and Q. He. Direct rendering of freeform volumes. *Computer-aided Design*, 27(7):553–558, 1995.
- [3] Alexandre Joel Chorin and Jerrold E Marsden. *A mathematical introduction to fluid mechanics*, volume 3. Springer, 1990.
- [4] J Austin Cottrell, Thomas JR Hughes, and Yuri Bazilevs. *Isogeometric analysis: toward integration of CAD and FEA*. John Wiley & Sons, 2009.
- [5] P. Deufhard and F. Bornemann. *Scientific computing with ordinary differential equations*, volume 42 of *Texts in Applied Mathematics*. Springer, 2002.
- [6] P. Deufhard and A. Hohmann. *Numerische Mathematik 1: Eine algorithmisch orientierte Einführung*. De Gruyter Lehrbuch Series. Walter De Gruyter Incorporated, 2008.
- [7] Christopher Dyken, Gernot Ziegler, Christian Theobalt, and Hans-Peter Seidel. High-speed marching cubes using histopyramids. *Comput. Graph. Forum*, 27(8):2028–2039, 2008.
- [8] Klaus Engel, Markus Hadwiger, Joe M Kniss, Christof Rezk-Salama, and Daniel Weiskopf. *Real-time volume graphics*. AK Peters, Limited, 2006.
- [9] Klaus Engel, Martin Kraus, and Thomas Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, HWWS '01, pages 9–16, New York, NY, USA, 2001. ACM.
- [10] Cass Everitt. Interactive order-independent transparency, 2001. NVIDIA white paper.
- [11] Jon Hjelmervik. Direct pixel-accurate rendering of smooth surfaces. In *Mathematical Methods for Curves and Surfaces, 2012*, volume 8177 of *Lecture Notes in Computer Science*, pages 238–247. Springer Berlin Heidelberg, 2014.
- [12] Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, Ltd., 2001.
- [13] James T. Kajiya. Ray tracing parametric patches. *SIGGRAPH Comput. Graph.*, 16(3):245–254, July 1982.
- [14] Yair Kurzion and Roni Yagel. Space deformation using ray deflectors. In *6th Eurographics Workshop on Rendering 95*, pages 21–32, 1995.
- [15] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics & Applications*, 8(3):29–37, May 1988.
- [16] Charles Loop and Jim Blinn. Real-time gpu rendering of piecewise algebraic surfaces. *ACM Trans. Graph.*, 25(3):664–670, July 2006.
- [17] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pages 163–169, New York, NY, USA, 1987. ACM.
- [18] T. Martin, E. Cohen, and M. M. Kirby. Direct isosurface visualization of hex-based high-order geometry and attribute representations. *IEEE Transactions on Visualization and Computer Graphics*, 18(5):753–766, 2012.
- [19] William Martin and Elaine Cohen. Representation and extraction of volumetric attributes using trivariate splines: A mathematical framework. In *Proceedings of the Sixth ACM Symposium on Solid Modeling and Applications*, SMA '01, pages 234–240, New York, NY, USA, 2001. ACM.
- [20] Kevin Myers and Louis Bavoil. Stencil routed A-buffer. In *ACM SIGGRAPH 2007 Sketches*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.
- [21] Lawrence Perko. *Differential Equations and Dynamical Systems*, volume 7. Springer, 2001.
- [22] Stefan Roettger, Stefan Guthe, Daniel Weiskopf, Thomas Ertl, and Wolfgang Strasser. Smart hardware-accelerated volume rendering. In *VisSym*, volume 3, pages 231–238. Citeseer, 2003.
- [23] Andre Schollmeyer and Bernd Fröhlich. Direct isosurface ray casting of nurbs-based isogeometric analysis. *IEEE Transactions on Visualization and Computer Graphics*, 20(9), 2014.
- [24] Michael Spivak. A comprehensive introduction to differential geometry, volume i. *Publish or perish*, Berkeley, 1979.
- [25] Jason C. Yang, Justin Hensley, Holger Grün, and Nicolas Thibieroz. Real-time concurrent linked list construction on the gpu. In *Proceedings of the 21st Eurographics Conference on Rendering*, EGSR'10, pages 1297–1304, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.
- [26] Young In Yeo, Lihan Bin, and Jörg Peters. Efficient pixel-accurate rendering of curved surfaces. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '12, pages 165–174, New York, NY, USA, 2012. ACM.



Franz G. Fuchs received his master's degree (Diplom) in mathematics from the Technical University of Munich (TUM) in 2006 with a thesis on image processing. In 2009 he received his PhD in applied mathematics from the University of Oslo (CMA), working on mathematical theory and numerical methods for hyperbolic conservation laws. His additional research interests include efficient numerical algorithms on parallel architectures for visualization and computation.



Jon M. Hjelmervik received his PhD in cotutelle between the University of Oslo and Grenoble INP in 2009. He is a research manager associated to SINTEF ICT Applied Mathematics since 1998. Until spring 2010 he held a 20% position as associate professor at Norwegian School of Information Technology (NITH). His research interests include visualization of isogeometric representations and heterogeneous computing in cloud-based frameworks.